Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 23, no. 2, pp. 293–316 (2019)
DOI: 10.7155/jgaa.00492

Constructing Hard Examples for Graph Isomorphism

Anuj Dawar *Kashif Khan*

Department of Computer Science and Technology
University of Cambridge

Abstract

We describe a method for generating graphs that provide difficult examples for practical Graph Isomorphism testers. We first give the theoretical construction, showing that we can have a family of graphs without any non-trivial automorphisms which also have high Weisfeiler-Leman dimension. The construction is based on properties of random 3XOR-formulas. We describe how to convert such a formula into a graph which has the desired properties with high probability. We validate the method by experimental implementations. We construct random formulas and validate them with a SAT solver to filter through suitable ones, and then convert them into graphs. Experimental results demonstrate that the resulting graphs do provide hard examples that match the hardest known benchmarks for graph isomorphism.

Submitted: September 2018
Reviewed: January 2019
Revised: March 2019
Accepted: March 2019
Final: April 2019
Published: April 2019
Article type: Regular Paper
Communicated by: G. Liotta

The research reported here was carried out while the second author was a student at the University of Cambridge during the academic year 2016–17. The research of the first author is supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1.

E-mail addresses: anuj.dawar@cl.cam.ac.uk (Anuj Dawar) kashif-raza-khan@hotmail.com (Kashif Khan)

1 Introduction

Graph Isomorphism (GI) is the problem of deciding, given two graphs G and H whether there is a bijection between their sets of vertices $V(G)$ and $V(H)$ respectively, that takes edges of G to edges of H and non-edges of G to non-edges in H. In short, it asks if G and H are the same, up to a renaming of vertices. The problem is of great interest in the field of complexity theory as it is among the few natural problems in NP that are not known to be in P nor known to be NP-complete. Babai's recent result [9] places the problem in *quasi-polynomial* time, further cementing its status as a candidate NP-*intermediate* problem.

While the complexity of GI is interesting from the theoretical standpoint, in practice the problem is largely solved. That is, there exist programs which efficiently deal with instances of graph isomorphism that arise in practice, for instance in searching through chemical databases or in image processing. Significant among these effective graph isomorphism testers are the programs `Traces` and `nauty`, available in a common distribution (see [20]). It remains a challenge for the theoretician to examine the algorithms behind these programs and determine their worst-case behaviour. Indeed, in the long version of his paper, Babai asks the question [8, Sec. 13.5]:

> The question is, does there exist an infinite family of pairs of graphs on which these heuristic algorithms fail to perform efficiently? The search for such pairs might turn up interesting families of graphs.

We address this question and provide a means of constructing just such a family of graphs, including an implementation and experimental results.

Computationally, the graph isomorphism problem is equivalent to the problem of determining the orbits of the automorphism group of a graph G. That is, given a graph G, we wish to partition $V(G)$ into the minimum number of classes so that for any pair of vertices u, v in the same class there is an automorphism of a graph G that takes u to v. We call this the *orbit partition* of the graph G. The fundamental algorithm underlying `nauty` as well as `Traces`, like many practical approaches to the graph isomorphism problem, relies on steadily refining a partition to arrive at the partition into orbits. It does this through a process of *(i) vertex refinement* combined with *(ii) individualization* and *(iii)* factoring of automorphisms of the graph. The process of vertex refinement, also known as colour refinement, is a highly efficient method that is known to achieve the orbit partition on *almost all* graphs [10] but fails on regular graphs, for example. Where vertex refinement fails, the programs use individualization, which is the process of selecting (i.e. individualizing) a particular vertex and placing it in its own class and then repeating vertex refinement until evenutually a partition into singleton sets is obtained. With backtracking, the structure of the parition into orbits is revealed. The main differences between the various graph isomorphism solvers (not only `nauty` and `Traces` but also `bliss` and `conauto`) are precisely in how the vertices are selected. This process is inherently exponential in the number of selections that need to be made. However, the search space is radically cut down if we can identify non-trivial automorphisms of the graph and factor the graph suitably, which Traces, in particular, does effectively. More details are given in Section 2 below.

In theory, the vertex refinement algorithm is subsumed by the k-dimensional Weisfeiler-Leman (k-WL) method, which works by refining a partition of the k-tuples of the vertices of a graph G (see [16, Sec. 2] for a good account of the history of this method). Taking k to be large enough (of the order of the number of vertices in G), the k-WL method gives exactly the orbit partition, but this is (for $k \geq 2$) not a practical method and rarely used in solvers. However, what is interesting from the point of view of worst-case analysis is that the k-WL method serves as a way of bounding the number of individualizations we need to determine the orbit partition in a graph G. To be precise, suppose G has k vertices $v_1, \dots, v_k$ such that when each of them is individualized, the vertex refinement procedure converges to the discrete partition of G, then we can also determine the orbit partition by the $(k + 2)$-WL method. Since it is known, through the construction of Cai, Fürer and Immerman [16] (called CFI graphs below), that there are, for every k, graphs on which the k-WL method fails to give the orbit partition, it follows that there is no constant bound on the number of individualizations needed, in combination with vertex refinement, to obtain the orbit partition of a graph. Hence, there is no polynomial bound on the running time of a graph isomorphism algorithm based solely on vertex refinement and individualization.

However, Traces has another element in its armoury, in that it detects automorphisms while constructing the orbit partition, using these to factor the graph and therefore cut down the search space. This means that the running time is not exponential in the number of individualizations but is potentially divided by the size of the automorphism group of the graph. Indeed, the CFI graphs are among the standard benchmarks considered in [20] and they prove to be not too difficult for the program to handle as they have many automorphisms. This led the first author of the present paper to suggest (at the December 2015 Dagstuhl seminar on Graph Isomorphism) that the way to construct hard families of graphs and answer Babai's quetsion, is to find graphs whose Weisfeiler-Leman dimension is large but which have no non-trivial automorphisms. A construction of structures satisfying this property, known as *multipedes* is given in the work of Gurevich and Shelah [17]. These structures can be easily turned into graphs to yield the desired family. However, while the theoretical construction guarantees the existence of such graphs, it turns out that constructing actual instances, even for small values of k, leads to very large graphs. Thus, in order to construct families that can be used for practical benchmarking of GI solvers, a refined analysis is required. One such approach was taken by Neuen and Schweitzer in [22] where the multipede construction was combined with *size reduction* techniques.

In the present paper, we give an alternative construction of such graphs which proves very effective. Instead of the multipedes of Gurevich and Shelah, we start with random systems of equations over the 2-element field. This is based on the insight from [4] that the construction of CFI graphs really codes such systems in the graph construction. We use a way of lifting these systems to graphs which have the property that as long as the original system has no non-trivial solutions, the resulting graph has no non-trivial automorphisms. Moreover, as long as the original system is k-locally satisfiable (a precise definition will follow), the orbit partition of the graph cannot be obtained by the l-WL method for some large l. It turns out that a random system has both properties: of having no non-trivial solutions and being k-locally satisfiable for sublinear values of k. The theoretical basis of this construction is given in Section 3. The main conclusion is Theorem 1 which shows that we can construct families of graphs which are *asymmetric*, i.e. have no non-trivial automorphisms, but have *linear* Weisfeiler-Leman dimension. Moreover, we define a randomized construction that produces such graphs with high probability.

It is instructive to compare our theoretical result with that of Neuen and Schweitzer [21, 22]. They also give a construction of families of graphs which provably require exponential time on a solver based only on individualization and refinement. Here, they take refinement to be any class of procedures that respect k-WL equivalences for some fixed k. Their construction is based on the multipede construction of Gurevich and Shelah, converted into graphs. This construction guarantees that the graphs are asymmetric, and have unbounded Weisfeiler-Leman dimension. While the dimension is unbounded, the multipede construction does not yield *linear* dimension. Indeed, as noted above, the graphs obtained for small values of k are rather large. Therefore, in order to obtain exponential lower bounds Neuen and Schweitzer employ size reduction techniques and explicitly consider the shape of the individualization search tree. By contrast, we directly establish a linear lower bound on the WL-dimension of the graphs and this immediately leads to an exponential lower bound on the search tree size under any target cell selection strategy. The experimental results reported in [21] are comparable with the ones we obtain.

To implement our theoretical construction, we leverage a highly developed SAT solver. This enables us to search for systems of equations which have no non-trivial solutions by coding them as 3SAT instances. While we do not directly check for k-local satisfiability, we use a proxy which is checking the speed improvement in the SAT solver that is obtained by the use of Gaussian elimination methods. This filter allows us to select those systems which are most likely to be locally satisfiable. We present details of the method in Section 4. Finally, we present some experimental results in Section 5 which show that the method does, indeed, yield instances which are hard, especially for `Traces`, but also for other isomorphism solvers. We have created some benchmark sets of such graphs, and one of these is now available through the webpage for `nauty` and `Traces`: `http://pallini.di.uniroma1.it/Graphs.html`. However, we consider the main contribution of the present work to be the method, using a SAT solver, which gives the ability to generate such large and hard example graphs at will.

The experimental work reported here was carried out in three stages. The first set of experiments were performed in April-June 2017 as part of the second author's Master's project. The full code of the implementation, all data generated in the experiments, as well as a project write-up can be found here:

https://github.com/kkcam/graph-ismorphism. In particular, a number of graphs in dreadnaut format can be downloaded from the site to run directly with nauty/Traces as well as in the DIMACS format for other solvers. A second set of experiments on the same graphs, involving a wider range of solvers, was carried out in January-February 2019, in response to a request from a referee. The results are available from the same website. Finally, another set of graphs using this protocol was created by Yui Chi Richie Yeung. Those graphs and the results are available at https://github.com/y-richie-y/sat_cfi/. A selection of results from all three sets of experiments is given in Section 5.

The theoretical work detailed in Section 3 is based on the first author's project suggestion, given as an appendix in the Master's thesis available at https://github.com/kkcam/graph-ismorphism.

2 Background

Partitions and Isomorphisms. Given a set S and two partitions $\mathcal{P} = \{P_1, \dots, P_s\}$ and $\mathcal{Q} = \{Q_1, \dots, Q_t\}$ of S, we say that $\mathcal{P}$ is a *refinement* of $\mathcal{Q}$ (or equivalently, that $\mathcal{Q}$ is *coarser* than $\mathcal{P}$) if for every $P \in \mathcal{P}$ there is a $Q \in \mathcal{Q}$ such that $P \subseteq Q$. It is a *proper refinement* if $s > t$. We say that a partition $\mathcal{P}$ is *discrete* if every part is a singleton.

We always consider undirected, loopless, simple graphs. That is, a graph G is a set of vertices $V(G)$ along with a set of edges $E(G)$ where each edge $e \in E(G)$ is a two-element set $e = \{u, v\} \subseteq V(G)$, with $u \neq v$. Where G is clear from context, we simply write V and E for the vertex and edge set respectively. For a set C, a C-coloured graph is a a graph G together with a function $\chi : V \to C$. For a C-coloured graph (G, χ), we refer to the partition of V given by $\{\{v \mid \chi(v) = c\} \mid c \in C\}$ as the χ-partition of V.

Given two C-coloured graphs (G, χ) and (H, δ), an *isomorphism* from the first to the second is a bijection $\iota : V(G) \to V(H)$ such that for each $u, v \in E(G)$, $\{u, v\} \in E(G)$ if, and only if, $\{\iota(u), \iota(v)\} \in E(H)$ and $\chi(v) = \delta(\iota(v))$. An *automorphism* of (G, χ) is an isomorphism from (G, χ) to itself. The *orbit partition* of (G, χ) is the coarsest partition of $V(G)$ such that if u and v are in the same part of the partition, there is an automorphism ι of (G, χ) such that $\iota(u) = v$. Consider the three computational problems: (1) given a pair of graphs G and H, decide if there is an isomorphism from G to H; (2) given a pair of coloured graphs (G, χ) and (H, δ) decide if there is an isomorphism between them; and (3) given a coloured graph (G, χ), output its orbit partition. It is known that these three problems are polynomially-equivalent, which is to say that there are polynomial-time reductions between any pair of them (see, for instance [24]). As such, we treat them as equivalent and mostly concentrate on the third.

A graph G, or a coloured graph (G, χ), is called *asymmetric* if its only automorphism is the identity map. Some authors call such graphs *rigid*, but we employ the terminology of Babai [7] who reserves the latter term for graphs without non-trivial endomorphisms.

Refinement and Individualization. The *vertex refinement* procedure is an algorithm that produces, given a C-coloured graph (G, χ), the coarsest partition $\mathcal{P}$ of V refining the χ-partition such that if $u, v \in P \in \mathcal{P}$, then for every $Q \in \mathcal{P}$, u and v have the same number of neighbours in Q. Note that the partition of V produced by the vertex refinement procedure is coarser than the orbit partition. For many graphs G it is, in fact, the orbit partition but, for example for regular graphs with no colouring, it can be properly coarser (see [10]).

Given a C-coloured graph (G, χ) and a vertex $v \in V$, let c be a new colour that does not appear in C and let $\chi' : V \to C \cup \{c\}$ be defined as the colouring with $\chi'(v) = c$ and $\chi'(u) = \chi(u)$ for $u \neq v$. Then, the partition of V obtained by vertex refinement from (G, χ') is called the vertex refinement of (G, χ) with *individualization* of v. More generally, given a sequence $I = (v_1, \dots, v_k)$ of vertices in V, the vertex refinement with individualizations of I is the algorithm that produces the vertex refinement of (G, χ') where χ' is a $C \cup \{c_1, \dots, c_k\}$-colouring of $V(G)$ with $\chi'(v_i) = c_i$ and $\chi'(v) = \chi(v)$ for $v \notin I$; and $c_1, \dots, c_k \notin C$. This partition is *not* in general coarser than the orbit partition. The aim of the individualization-refinement procedure is to find the smallest (in a precise sense) I such that the refinement with individualizations of I yields a partition into singleton sets. From this, it is possible to produce the orbit partition of (G, χ). For details, we refer the reader to [20, Sec. 2].

High-dimensional Weisfeiler-Leman. The k-dimensional Weisfeiler-Leman algorithm, for any $k \geq 2$, gives a partition of V^k that is the coarsest partition $\mathcal{P}$ satisfying the following stability condition: if $\mathbf{u}, \mathbf{v} \in V^k$ are tuples in the same part of $\mathcal{P}$ and $(P_1, \dots, P_k)$ is a k-tuple of parts of $\mathcal{P}$, then the order-preserving map from $\mathbf{u}$ to $\mathbf{v}$ is an isomorphism of the induced subgraphs of G and $|\{x \mid \mathbf{u}[x/i] \in P_i \text{ for } 1 \leq i \leq k\}| = |\{x \mid \mathbf{v}[x/i] \in P_i \text{ for } 1 \leq i \leq k\}|$. Here, $\mathbf{u}[x/i]$ denotes the tuple obtained by substituting x for the ith element of $\mathbf{u}$ and $|S|$ denotes the cardinality of a set S. We write $\equiv^k$ to denote the equivalence relation corresponding to this partition. Again, this partition is necessarily coarser than the partition of V^k into orbits under the action of the automorphism group of G, since the orbit partition clearly satisfies the stability condition. Also, for sufficiently large values of k, in particular certainly for $k \geq n - 1$, the partition given by $\equiv^k$ *is* the orbit partition. So, for any graph G, we define the *Weisfeiler-Leman dimension* of G, denoted $\mathrm{WL}(G)$, to be the least k such that the partition induced by $\equiv^k$ on V^k coincides with the orbit partition.

Cai, Fürer and Immerman [16] showed that there is no fixed k such that $\mathrm{WL}(G) < k$ for all graphs G. Indeed, they show a linear lower bound on $\mathrm{WL}(G)$. To be precise, they construct for each k a pair G and H of non-isomorphic graphs with $O(k)$ vertices such that G and H cannot be distinguished by the isomorphism test based on k-dimensional Weisfeiler-Leman equivalence. This implies that $\mathrm{WL}(G \uplus H) > k$, where $G \uplus H$ is the disjoint union of G and H. Through this operation of disjoint union, the problem of testing graph isomorphism reduces to that of constructing the orbits of the automorphism group and, as stated in the introduction, here we adopt the latter perspective.

The k-dimensional Weisfeiler-Leman isomorphism test has been extensively analyzed in theoretical studies of the graph isomorphism problem. It has many equivalent and strikingly different characterizations, arising in algebra, combinatorics, logic and optimization. In particular, it is known that in a graph G, $\mathbf{u} \equiv^k \mathbf{v}$ if, and only if, there is no formula of C^{k+1} (first-order logic with counting quantifiers using at most $k+1$ distinct variables) that distinguishes $\mathbf{u}$ from $\mathbf{v}$. This relation was given a useful characterization in terms of a k-pebble *bijective game* by Hella [18]. The game is played by two players called Spoiler and Duplicator on a graph G. The position of the game at any moment in time consists of two k-tuples of vertices $\mathbf{u}$ and $\mathbf{v}$. In each move, Spoiler chooses a value of $i \in [k]$ and Duplicator responds with a bijection $\pi : V \to V$. Spoiler then chooses a vertex $x \in V$ and the new position is $\mathbf{u}[x/i]$ and $\mathbf{v}[\pi(x)/i]$. Spoiler wins in any position if the *ordered* subgraph of G induced by the tuple $\mathbf{u}$ is not isomorphic to the *ordered* subgraph induced by $\mathbf{v}$. The result of Hella [18] essentially tells us that Duplicator has a strategy to play forever without Spoiler winning in the $(k+1)$-pebble bijective game starting at position $\mathbf{u}, \mathbf{v}$ if, and only if, $\mathbf{u} \equiv^k \mathbf{v}$.

The connection between the Weisfeiler-Leman dimension of a graph and the refinement and individualization procedure is the following. If a graph G contains k vertices $v_1, \ldots, v_k$ such that individualizing them results in the vertex refinement procedure on G producing the discrete partition, then $\mathrm{WL}(G) \leq k+2$. This is most easily seen through the characterization of the Weisfeiler-Leman dimension in terms of counting logic. That is, when such a set of k vertices exist, we can show that any pair x and y of vertices that are not distinguished by any formula of C^{k+3} are in the same orbit of the automorphism group of G. The argument is as follows. We know that the vertex refinement procedure yields a partition into C^2-equivalence classes. By the assumption that individualizing $v_1, \ldots, v_k$ results in the discrete partition, we have that for each vertex x, there is a formula $\phi(x)$ of C^2 using constants for $v_1, \ldots, v_k$ that is only true of x and of no other vertex in G. Write ϕ_x for the formula of C^{k+2} where the constants in ϕ have been replaced with new variables (which we will also denote $v_1, \ldots, v_k$ for convenience). Now consider the formula

$$\theta(v_1, \ldots, v_k) := \bigwedge_{x \in V(G)} \exists^{=1} x \phi_x \wedge \forall x, y \big(E(x,y) \leftrightarrow \bigvee_{\{x,y\} \in E(G)} (\phi_x \wedge \phi_y)\big).$$

Note, here, while we have a different formula ϕ_x for each $x \in V(G)$, we assume that we re-use the variable x. However, where we write ϕ_y, we replace it with the new variable y. This ensures that the total number of variables used is at most $k+3$. It can then be verified that the formula $\exists v_1, \ldots, v_k \theta$ describes the graph G uniquely, up to isomorphism. Moreover, $\exists v_1, \ldots, v_k(\theta \wedge \phi_x)$ is a formula of C^{k+3} that is only satisfied in G by vertices in the orbit of x.

Multipedes. Gurevich and Shelah [17] show how to construct a class of finite structures which is (1) axiomatized by a sentence of first-order logic; (2) contains only structures with no non-trivial automorphisms; and (3) such that no formula of fixed-point logic defines a linear order on all structures in this class. From our point of view, the relevant aspect of this construction is that it gives, for each value of k, an *asymmetric* structure M_k, that is one which has no non-trivial automorphisms, but on which the partition into $\equiv^k$-classes is non-trivial. That is to say, even though every element of the orbit partition of M_k is a singleton, there are pairs of distinct elements a, b in M_k such that $a \equiv^k b$. The structures in question are called 3-multipedes in [17].

It seems at first sight that this provides suitable hard examples for graph isomorphism testers such as `Traces`. Strictly speaking, the 3-multipedes are not graphs, but they can be translated to graphs by standard methods (see [19, Theorem 5.5.1]), preserving the relevant properties: unbounded Weisfeiler-Leman dimension and no non-trivial automorphisms. The absence of non-trivial automorphisms means that factoring by automorphisms cannot be used to speed up search by trimming the tree, while the unbounded dimension guarantees that there is no upper bound on the number of individualization steps needed to make the vertex refinement procedure yield the discrete partition. However, actually constructing instances of such multipedes turns out to be difficult. The proof in [17] does not actually show how to construct the structures M_k. Rather, it shows that for all large enough values of n, a random structure on n elements, under a suitable skew probability distribution μ_n, has the right properties. However, the probability grows rather slowly with n. Indeed, the smallest value of n at which the probability is non-zero is possibly exponential in k. An experimental attempt to sample from the distribution μ_n failed to produce interesting examples at values of n up to a few thousand [2].

In the present paper, we consider an alternative construction, based on similar principles, of graphs whose Weisfeiler-Leman dimension is linear in the number of vertices (as with the CFI graphs), and which have no non-trivial automorphisms. The construction is again randomized, but based on a simple and well-understood probability distribution. Furthermore, a use of a SAT solver enables the quick generation of examples by filtering graphs that are sampled from the distribution.

3 The Construction

In this section, we describe a construction that yields, for each k, a graph G_k with $O(k)$ vertices with the property that G_k is asymmetric and has Weisfeiler-Leman dimension at least k. The proof that such graphs exist is derived from known results in the literature, and here we show how to derive it, giving the necessary definitions to understand the background. The starting point of the construction is the observation that we can define instances of 3-XOR that are k-locally consistent but unsatisfiable.

XOR formulas. Fix a countable set $\mathcal{X}$ of Boolean variables. We use capital letters $X, Y, \ldots$ to range over this set. A 3-XOR-formula is a finite set of clauses, where each clause contains exactly 3 literals, each of which is either a variable X or a negated variable $\bar{X}$.

We say that a 3-XOR-formula ϕ is satisfiable if there is an assignment $T : \mathcal{X} \to \{0,1\}$ of truth values to the variables $\mathcal{X}$ such that in each clause of ϕ, an *even* number of literals is made true.

Given a 3-XOR-formula ϕ, we can construct a system of linear equations over the two-element field $\mathbb{F}_2$. That is, for each clause C of ϕ we construct the equation $x + y + z = c$ where x, y, z are the variables occurring in the literals of C and c is 1 if an odd number of them appear negated and 0 otherwise. It is easily verified that this system of equations has a solution if, and only if, ϕ is satisfiable. Note that two distinct clauses may give rise to the same equation. Say that two clauses are *equivalent* if they give rise to the same equation. In the sequel, we will use the terminology of 3-XOR formulas and of systems of linear equations interchangeably, according to which is convenient.

So, we can think of a 3-XOR formula with n variables and m clauses as a system of equations $H\mathbf{x} = \mathbf{b}$ where H is an $m \times n$ matrix, $\mathbf{x}$ the tuple of n variables and $\mathbf{b} \in \mathbb{F}_2^m$ the m-tuple of right-hand sides of the equation. We say the system is *homogeneous* if the right-hand side of every equation is 0. This corresponds to a 3-XOR formula in which no variable appears negated. A homogeneous system is always satisfiable, as it is satisfied by the assignment of 0 to every variable. Note that a homogeneous system is completely specified by a collection of 3-element sets of variables, with one set for each equation, containing the three variables that appear in it.

Random XOR formula. For fixed positive integers m, n we write $F(m,n)$ for the set of all 3-XOR-formulas over the variables $X_1, \dots, X_n$ containing exactly m inequivalent clauses. We also write $\mathcal{F}(m,n)$ for the uniform probability distribution over $F(m,n)$. It is known that, for large enough values of m and n, with $m > n$, a random formula drawn from this distribution is unsatisfiable (see [23]). That is to say that as n increases, for all $m > n$, the probability that a formula drawn from the distribution $\mathcal{F}(m,n)$ is satisfiable tends to 0.

We are interested in 3-XOR-formulas that are unsatisfiable but k-locally consistent, for suitable integer k. For our purposes, we define k-local consistency by means of the following pebble game, played by two players called Spoiler and Verifier. The game is played on a 3-XOR-formula ϕ with k pebbles $p_1, \dots, p_k$. At each move Spoiler chooses a pebble p_i (either one that is already in play, or a fresh one) and places it on a variable X appearing in ϕ. In response, Verfier has to choose a value from $\{0,1\}$ for the variable X. If, as a result, there is a clause C such that all literals in C have pebbles on them and the assignment of values to them given by Verifier results in C being unsatisfied, then Spoiler has won the game. Otherwise the game can continue. If Verifier has a strategy to play the game forever without losing, we say that ϕ is k-locally consistent.

It is also known that for all k, the probability that a random formula drawn from $\mathcal{F}(m,n)$ (with $m > n$) is k-locally consistent tends to 1 as n increases. This was proved for formulas of 3SAT rather than 3-XOR in [3], but a similar analysis shows the result also for 3-XOR. Such an analysis can be found in [5, Lemma 4].

Unique Satisfiability. As noted above, a homogeneous system of equations is always satisfiable, as it is satisfied by the assignment of 0 to every variable. We say that the system is *uniquely satisfiable* if this is the only solution to the system. It is easy to see that the set of solutions to $H\mathbf{x} = 0$ is exactly the null space of the matrix H, as a subset of the vector space $\mathbb{F}_2^n$. In particular, the system is uniquely satisfiable if, and only if, H has rank n.

Define $H(m, n)$ to be the set of all homogeneous systems of equations with m clauses and n variables, and $\mathcal{H}(m, n)$ for the uniform probability distribution over this set. We use the following fact about this distribution, established in [11]

Lemma 1 *There is a threshold $t > 1$ such that, for any $\alpha > t$, the probability that a random system drawn from $\mathcal{H}(\alpha n, n)$ is uniquely satisfiable tends to* 1 *as n increases.*

A homogeneous system is necessarily satisfiable, and so also k-locally consistent for all k. If it is uniquely satisfiable, it has no solutions when we require some fixed variable X_i to take value 1. However, for a randomly chosen such system, adding the condition $X_i = 1$ leaves it k-locally consistent for small values of k. To be precise, there is a constant γ such that if ϕ is a system $H\mathbf{x} = 0$ chosen at random from $\mathcal{H}(\alpha n, n)$, then with high probability there is some i such that if ϕ_i is the system obtained from ϕ by adding the equation $X_i = 1$ then ϕ_i is γn-locally consistent. This is a much weaker statement than proved in [5, Lemma 4] where it is shown that changing the right-hand sides of a random subset of the vertices to 1 still leaves the system γn-locally consistent.

CFI construction The construction of Cai, Fürer and Immerman [16] provides us with examples of pairs of non-isomorphic graphs which are not distinguished by the k-dimensional Weisfeiler-Leman test. Inspired by this, a construction described in [1, Prop. 32] shows how to convert any k-locally consistent system of equations in $H(m, n)$ to one that cannot be distinguished by the k-dimensional Weisfeiler-Leman test from its homogeneous companion. Here, the *homogeneous companion* of a system $H\mathbf{x} = \mathbf{b}$ is $H\mathbf{x} = \mathbf{0}$ (see also [5, Lemma 3] for a similar argument). Here we adapt the construction, to turn an arbitrary system ϕ into a graph G_ϕ with the property that the local consistency of ϕ translates into a lower bound on the Weisfeiler-Leman dimension of G_ϕ. Moreover, the *unique satisfiability* of ϕ guarantees that G_ϕ is asymmetric.

For any 3-XOR-formula ϕ, we define the graph G_ϕ by the following construction. If ϕ has m inequivalent clauses and n variables, G_ϕ has a total of $4m + 2n + 3(n - 1)$ vertices.

Let $X_1, \ldots, X_n$ be the n variables in some fixed order. For each clause C of ϕ, we define the four clauses $C_{000}, C_{011}, C_{110}$, and C_{101} by letting $C_{000} = C$ and $C_{011}, C_{110}, C_{101}$ be the three clauses equivalent to C obtained by negating exactly two of the literals of C. In particular, C_{011} is obtained from C_{000} by negating the second and third variables in the clause, C_{110} by negating the first and second and C_{101} by negating the first and third. Here, the terms "first", "second" and "third" refer to the numberical order of the variables chosen above.

We then have a vertex in G_ϕ for each of these clauses. Also, for each variable X in ϕ, we have two vertices X^0 and X^1. In addition, for each i with $1 \leq i < n$ we have three vertices i_l, i_r, i_s.

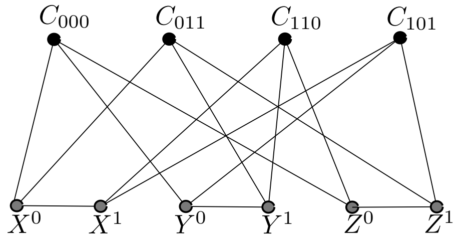

Figure 1: Clause gadget in G_ϕ corresponding to the clause $X \oplus Y \oplus Z$

The edges are described as follows. For each clause C, if the literal X occurs in C, we have an edge from C to X^1 and if the literal $\bar{X}$ occurs in C, we have an edge from C to X^0. There is an edge between X^0 and X^1. These are depicted in Figure 1. These capture the essence of the CFI-like construction. In addition, for each i we also have the edges: (i_l, i_r), (i_r, i_s) and (i_l, X_i^0), (i_l, X_i^1), (i_r, X_{i+1}^0) and (i_r, X_{i+1}^1). These additional edges are depicted in Figure 2. The purpose of the additional gadget involving the vertices i_l, i_r and i_s is to remove some symmetries on the graph by imposing the chosen order on the set of variables.

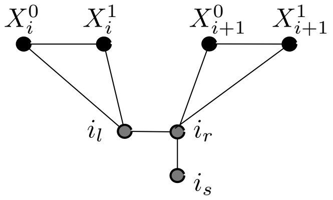

Figure 2: Asymmetry gadget in G_ϕ

Now, fix a homogeneous system of equations ϕ, and let G_ϕ be the graph obtained by the above construction. Also, let ϕ_i be the system obtained by adding the equation $X_i = 1$ to ϕ.

Lemma 2 *If ϕ_i is $3(k+1)$-locally consistent, then $X_i^0 \equiv^k X_i^1$ in G_ϕ.*

Proof: The proof follows along the lines of [5, Lemma 3] by showing a winning strategy for Duplicator in the bijective $(k+1)$-pebble game played on G_ϕ starting in the position $\mathbf{u}, \mathbf{v}$ where $\mathbf{u}$ is the tuple consisting of the vertex X_i^0 repeated $k+1$ times and $\mathbf{v}$ is the tuple consisting of the vertex X_i^1 repeated $k+1$ times. We give a brief outline.

Duplicator's strategy will always be to play a bijection that is the identity on the vertices i_l, i_r, i_s. For each variable X it maps the set $\{X^0, X^1\}$ to itself (though it may swap these two vertices) and for each clause C it maps the set $\{C_{000}, C_{011}, C_{110}, C_{101}\}$ to itself (though it may permute these elements). Moreover the permutation induced on $\{C_{000}, C_{011}, C_{110}, C_{101}\}$ must be either the identity or a permutation induced by swapping X^0 and X^1 for exactly two variables X appearing in the clause C. Call a bijection meeting these requirements *well-defined.*

Given a position $\mathbf{u}, \mathbf{v}$ in the bijective game, we say that it is *consistent* if there is a well-defined bijection β taking $\mathbf{u}$ to $\mathbf{v}$ and such that for any $C_i \in \mathbf{u}$ if $\beta(C_i) = C_j$ where C_j is obtained from C_i by swapping X^0, X^1 and Y^0, Y^1 then $\beta(X^0) = X^1$ if either of X^0 or X^1 is in $\mathbf{u}$ and similarly $\beta(Y^0) = Y^1$ if either of Y^0 or Y^1 is in $\mathbf{u}$.

Consider now a consistent position $\mathbf{u}, \mathbf{v}$ and let β be a well-defined bijection witnessing this. Let U be the set of variables of ϕ containing all variables X such that either X^0 or X^1 appears in $\mathbf{u}$ or X appears in some clause C such that one of $C_{000}, C_{011}, C_{110}, C_{101}$ appears in $\mathbf{u}$. Note that U has at most $3(k+1)$ elements. Now, we define the map $T : U \to \{0, 1\}$ given by $T(X) = 0$ if $\beta(X^0) = X^0$ and $T(X) = 1$ if $\beta(X^0) = X^1$. Duplicator's strategy is to ensure that T is a winning position for Verifier in the $3(k+1)$-local consistency game on the formula ϕ. That is to say, starting in the position where pebbles are placed on the variables in U, and Verifier's responses are given by T, Verifier can continue and play forever.

This condition is satisfied by the initial position, as $\mathbf{u}$ is just the element X_i^0 repeated $k+1$ times, so $U = \{X_i\}$, and T is the map taking X_i to 1. But, the fact that ϕ_i is $k+1$-locally consistent implies that this is a winning position for Verifier. Now, to see that the Duplicator can maintain the position, suppose at some stage in the bijective game, Spoiler moves pebble j. Duplicator needs to construct a well-defined bijection such that anywhere Spoiler places the pebble will result in a consistent position. Spoiler's move can be translated into a move in the local consistency game from the current position T. Duplicator's possible responses in that game define a function from the variables X to $\{0, 1\}$ and this can be turned into a well-defined bijection. □

Asymmetry Finally, we want to argue that if the homogeneous system ϕ is uniquely satisfiable, then G_ϕ is asymmetric. Before giving the proof, we give some intuition. The graph G_ϕ has two vertices X^0 and X^1 for each variable X of ϕ. Consider first a permutation π of these vertices which fixes each set $\{X^0, X^1\}$. This gives rise to a map T from the variables of ϕ to $\{0, 1\}$ such that $T(X) = 1$ if, and only if, π exchanges X^0 and X^1. To extend π to an automorphism of G_ϕ would require us to permute the vertices corresponding to clauses in such way that fixed each set $\{C_{000}, C_{011}, C_{110}, C_{101}\}$. This can only happen if for exactly two of the variables X appearing in the clause C do we have $T(X) = 1$. In other words, this requires T to be an assignment satisfying ϕ. Are there other automorphisms of G_ϕ that do not fix the sets $\{X^0, X^1\}$? The presence of the vertices i_l, i_r, i_s ensures that these sets are effectively linearly ordered and no other automorphisms are possible. This is formally proved below.

Lemma 3 *If ϕ is homogeneous, then it is uniquely satisfiable if, and only if, G_ϕ is asymmetric.*

Proof: Let α be any automorphism of G_ϕ. Note that every clause vertex C has degree 3. Every variable vertex X^0 or X^1 has degree at least 4. Every vertex i_s has degree 1. Thus, each of the following sets is fixed (set-wise) by α:

- the set $S = \{i_s \mid 1 \leq i < n\}$: this is the set of vertices of degree 1;
- the set $R = \{i_r \mid 1 \leq i < n\}$: this is the set of vertices adjacent to a vertex in S;
- the set of clause vertices $\mathcal{C}$: this is the set of vertices of degree 3 that are not within distance 2 of a vertex in S;
- the set of variable vertices $\mathcal{X}$: this is the set of neighbours of $\mathcal{C}$; and
- the set $L = \{i_l \mid 1 \leq i < n\}$: this is everything else.

Indeed, we can say more. Each of the sets S, L and R is fixed *pointwise* by α. If this were not so, there would be some i, j with $i < j$ such that $\alpha(i_s) = j_s$ (since the set S is fixed). Then, $\alpha(i_r) = j_r$ (since these are the sole neighbours), $\alpha(\{X_i^0, X_i^1\}) = \{X_j^0, X_j^1\}$ (since these are the only neighbours in $\mathcal{X}$ of i_r and j_r respectively), and so $\alpha((i+1)_l) = (j+1)_l$ and $\alpha((i+1)_r) = (j+1)_r$. Proceeding by induction, we have for all k $\alpha((i+k)_r) = (j+k)_r$. Taking k large enough so that $j + k > n \geq i + k$, we get a contradiction.

It also follows that, for each variable X, $\alpha(\{X^0, X^1\}) = \{X^0, X^1\}$. That is, α either fixes each of the two vertices or it interchanges them. Note that if α fixes all the variable vertices, then it is the identity everywhere, since no two vertices in $\mathcal{C}$ have the same neighbours in $\mathcal{X}$. Let T be the assignment that maps X to 0 if α is the identity on $\{X^0, X^1\}$ and 1 otherwise. We can check that T satisfies ϕ.

In the other direction, suppose there is a truth assignment T that satisfies ϕ. Now consider the map on $\mathcal{X}$ that exchanges the vertices X^0 and X^1 just in case $T(X) = 1$ and is the identity everywhere else. We extend this to a map on $\mathcal{C}$ as follows. For any clause C of ϕ, there are either 0 or 2 variables X in C for which $T(X) = 1$, since T is a satisfying assignment. In the first case, we let our map be the identity on $\{C_{000}, C_{011}, C_{110}, C_{101}\}$ and in the latter case it is the unique permutation of this set induced by exchanging X^0 and X^1 for the two variables such that $T(X) = 1$. Finally, we also define the map to be the identity on the set $L \cup R \cup S$. It is now easy to see that this map is an automorphism. □

We can conclude the description of the construction with the statement of a theorem.

Theorem 1 *There is a family of asymmetric graphs G_k with $O(k)$ vertices and Weisfeiler-Leman dimension k.*

Proof: By Lemma 1, there is an α such that for sufficiently large n, a randomly chosen homogeneous system of equations ϕ from $\mathcal{H}(\alpha n, n)$ is uniquely satisfiable with positive probability. By Lemma 3, G_ϕ is then asymmetric. On the other hand, by [5, Lemma 5], ϕ_i is γn-locally consistent for any i with high probability, which implies by Lemma 2 that G_ϕ has WL-dimension at least $\frac{1}{3}\gamma n - 1$. □

Note that we really have proved, not only the existence of such a family, we have described a random process that produces such graphs with high probability.

Size Bounds While Theorem 1 tells us that the graphs G_k have size linear in k, the actual size bounds are somewhat less clear. Specifically, there is a probabilistic element to the construction that relies on constructing a uniquely satisfiable formula ϕ such that for some i, ϕ_i is k-locally consistent. What we know is that for any k, and any large enough n, a randomly constructed formula with n variables and $m = \alpha n$ clauses ($\alpha > 1$) will have these properties with high probability. How big does n have to be before the probability becomes significant? A direct calculation does not give much cause for optimism.

Our argument for why a random formula is k-locally consistent with high probability is based on [5, Lemma 3], which in turn relies on the argument for expansion of a random 3-uniform hypergraph given in [13]. The key combinatorial bound in on width is [13][Lemma 6.6], attributed to [12]. A simple calculation shows that we need m to be around 10^7 in order to be guaranteed a width lower bound of 2 (i.e. that a formula will be 2-locally consistent with probability greater than 1/2). With m around 10^9, we get reasonably high bounds on width, but these would be much larger graphs than we wish to consider. What we show in the rest of the paper is experimental results which show that even for much smaller values of n and m, a random sample produces graphs that are difficult for isomorphism testers. We combine random generation of 3-XOR formulas with a filtering process which is aimed at improving the likelihood of getting locally-consistent instances. We describe this in the next section.

4 Experimental Setup

Section 3 established a theoretical result that shows that a random graph constructed in a particular way has the properties of being asymmetric and having high Weisfeiler-Leman dimension. In outline, we want to start with a random homogeneous 3-XOR formula on n variables with αn clauses, i.e. a random 3-uniform hypergraph on n vertices and convert it into the graph G_ϕ. This graph is asymmetric if ϕ is uniquely satisfiable (which occurs with high probability). Moreover G_ϕ has Weisfeiler-Leman dimension at least k if ϕ is $3k$-locally satisfiable, an event that also occurs with high probability. We now describe an experimental setup for producing such graphs (with up to a few thousand nodes) by starting with a random formula and applying a succession of filters. In the process we apply a number of heuristics in addition to the theoretical approach outlined above. To motivate and justify these heuristics, we now break up the construction in a slightly different way.

Asymmetry Consider ϕ, a homogeneous 3-XOR formula with n variables $\mathcal{X} = \{X_1, \dots, X_n\}$ and m clauses $\mathcal{C} = \{C_1, \dots, C_m\}$. We identify this system with a bipartite graph Φ with vertices $\mathcal{C}$ on the left and $\mathcal{X}$ on the right and an edge between X and C if X appears in the clause C. Note that because ϕ is homogeneous, the graph determines ϕ completely. The construction of the graph G_ϕ described above can now be broken up into two steps. In the first step, we produce a graph G^1_ϕ by replacing each $X \in \mathcal{X}$ by two vertices X^0 and X^1 and each $C \in \mathcal{C}$ with four vertices $\{C_{000}, C_{011}, C_{110}, C_{101}\}$ and connecting them as described above and illustrated in Figure 1. In the second step, we augment the graph G^1_ϕ with additional vertices i_l, i_r, i_s for $i \in \{1, \dots, n\}$.

Note that the reason for the second step is effectively to impose a linear order on the set of variables $\mathcal{X}$ and thereby ensure that the only automorphisms of G_ϕ are the ones generated by satisfying assignments to ϕ. Thus, if the graph Φ is itself asymmetric, the second step is unnecessary as it is easily seen that in this case the only automorphisms of G^1_ϕ are generated from a satisfying truth assignment to ϕ by swapping X^0 and X^1 for all variables that are set to `true`. How likely is it that a random Φ (i.e. a random left-3-regular bipartite graph with m nodes on the left and n on the right) is asymmetric?

We can think of Φ as a 3-uniform hypergraph on n nodes, with m edges. It is easy to show (using the same methods that show that a random graph is asymmetric (see [15, Chap. 9])) that if m is roughly $n \log n$, then a random 3-uniform hypergraph is, indeed, asymmetric with probability going to 1 as n increases. This is not the case when $m = \alpha n$ for constant α. However, our experiments show that in the range of values of n we worked with (n up to about 3000, and m between 1 and 5), there was a reasonably high probability of coming up with an asymmetric hypergraph. Moreover, if Φ is asymmetric, this is reasonably quick to check with a tester such as `nauty/Traces` as it is also highly probable that vertex refinement gives the orbit partition. It is only when Φ is converted to G^1_ϕ that we get high Weisfeiler-Leman dimension. Thus, for the purpose of the experiments, instead of generating the graphs G_ϕ from Φ, we run a test on Φ to check if it has any non-trivial automorphisms. If it does, we discard it. Otherwise, we construct the graph G^1_ϕ and use that.

Unique Satisfiability Having generated a random homogeneous formula ϕ, we wish to check that it is uniquely satisfiable. For this, we use a highly developed SAT solver (`CryptoMiniSat 5`). This SAT solver is specifically optimized for cryptographic applications where the input often contains clauses that are formed by taking the XOR, rather than the disjunction, of a set of literals. `CryptoMiniSat` combines standard SAT solving methods (based on DPLL) with the selective use of Gaussian elimination to attack such problems quickly.

In our filter, we express ϕ as a conjunction of clauses where each clause is the XOR of three variables. We then test the satisfiability of $\phi' \equiv \phi \wedge \bigvee_{X \in \mathcal{X}} X$. That is, we add a clause that is the disjunction of all variables in $\mathcal{X}$. Of course, ϕ' is satisfiable if, and only if, ϕ has a satisfying assignment other than the all zeroes solution. In other words, ϕ' is satisfiable if, and only if, ϕ is *not* uniquely satisfiable.

Local Consistency We also want to ensure that the ϕ we select is k-locally consistent for a sufficiently large value of k. This is difficult to check directly. The problem of checking k-local consistency is known to be hard, requiring time exponential in k (see [14]) and we do not know of any good implementations. Instead, we used a simple heuristic that leverages the specific capabilities of `CryptoMiniSat`. Specifically, this package allows us to turn the use of Gaussian elimination on and off with an option. We check the satisfiability of the formula ϕ' by running `CryptoMiniSat` twice, once with Gaussian elimination on and once with it off. If the former is significantly faster than the latter, we expect that the ϕ we have is a good candidate. Note, however, that this does not give us any bounds on the value of k for which ϕ might be k-locally consistent.

To justify this heuristic, note that the DPLL methods (with clause-learning and restarts) as employed in modern SAT solvers are subsumed by bounded-width resolution (see [6]). And formulas that are highly locally consistent but not globally consistent are exactly the ones that are difficult for bounded-width resolution [3]. On the other hand, Gaussian elimination is a method that specifically is fast for solving systems of linear equations which may well be locally consistent [4]. Thus, a formula on which Gaussian elimination is quick to determine satisfiability but DPLL-based methods are slow is a prime candidate for us.

Summary of Methodology In summary, our methodology for generating hard examples for isomorphism testers is the following.

1. For a fixed value of n and m (roughly about $2n$), take a set $\mathcal{X}$ of n variables.

2. Randomly select m 3-element subsets of $\mathcal{X}$ to form the left-3-regular bipartite graph Φ.

3. Check (using `Traces`) to see if Φ has any non-trivial automorphisms. If so, discard it.

4. If Φ has no non-trivial automorphisms, form the formula ϕ' by taking the conjunction of the clauses $\bigoplus_{X \sim C} X$ for each left-node C of Φ along with the clause $\bigvee_{X \in \mathcal{X}} X$.

5. Check if the formula ϕ' is satisfiable using the SAT solver `CryptoMiniSat` with Gaussian elimination option on. If it is satisfiable, discard Φ.

6. Run `CryptoMiniSat` on ϕ' a second time, with the Gaussian elimination option turned off. If this takes significantly longer to determine ϕ' is unsatisfiable, then Φ is a prime candidate for the construction.

7. From Φ, obtain the graph G^1_ϕ by replacing each node C on the left-hand side with four nodes, and each node X in the right-hand side with two nodes and connecting them as described earlier.

5 Experimental Results

We can report on three sets of experimental results, using the construction described in the previous section. It should be noted that the main parameter that can be varied in the construction is the ratio m/n where m is the number of clauses and n the number of variables in the 3-XOR formula. The ratio needs to be at least 1 to guarantee that the constructed formula is uniquely satisfiable. The closer it is to 1, the less likely it is to be uniquely satisfiable. Indeed, experimental runs show that at smaller values we had to sample from the distribution $\mathcal{H}(m, n)$ many times over to find a uniquely satisfiable instance. On the other hand, the larger the value of m/n, the harder it is to find locally satisfiable instances. While the theoretical results guarantee that the probability of finding such instances increases with n, it clearly does so more slowly for large values of m/n. Hence, in practice, one needs to fine tune the right value of the ratio to get good results.

It should also be noted that our construction does not determine the actual WL-dimension of the graphs. This seems to be a much harder computational problem than testing isomorphism itself (see [14] for bounds on the related problem of determining k-local consistency). Thus, while the heuristic filters we use are *likely* to produce graphs of large WL-dimension, we are unable to actually state the dimension of the graphs produced.

First Set. The procedure for constructing graphs described in the previous section was run on a cloud server, with the specification given in Table 1, during April-June 2017. The results show that graphs of a few hundred nodes produced using this procedure are very difficult for `Traces` in the sense that in most cases, at this size, the system timed out (with a timeout set at 3 hours) and failed to identify the automorphism orbits.

Table 1: Test Environment 1

Feature	Description
Host	DigitalOcean
Operating System	Ubuntu 16.10 64-bit
Memory	2GB
Disk	20GB SSD
CPU	2 CPUs

Some results of test runs on graphs produced by our construction are shown in Figure 3. These plot the time taken to run `Traces` on graphs with n nodes

(n being the horizontal axis). The plot on the left of the figure is for graphs produced from 3-XOR formulas with n variables and $m = n$ clauses. The plot on the top right gives similar times for graphs produced from 3-XOR formulas with $m = 2n$. Here, virtually all graphs we were able to produce with over 5000 nodes timed out on `Traces`.

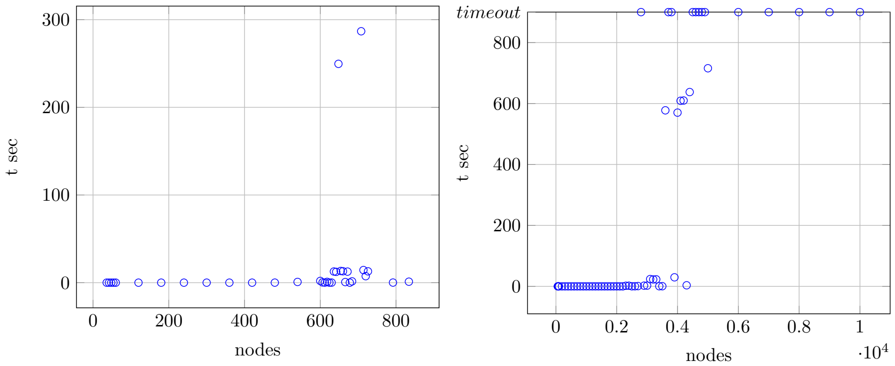

Figure 3: Left: n=m (con_n).
Right: n=2m (con_2n).

The complete data, including the graphs constructued, from this set of experiments is available at `https://github.com/kkcam/graph-ismorphism`. Some explanation of the nomenclature might be helpful. The graphs are classified according to the ratio m/n used in their construction. For instance `con_2n` is the collection of graphs with $m = 2n$, and `con_n` is the collection of graphs with $m = n$. There is also a package `con_sml` which contains for each n the graph with the smallest ratio m/n for which we were able to obtain a uniquely satisfiable 3-XOR formula, which also gives an asymmetric bipartite graph Φ.

Second Set In February 2019, we ran a second set of experiments. These used the same database of graphs produced by the construction for the first set. This time the virtual machine setup was as described in Table 2. Again, with `Traces`, most of the larger graphs timed out. However, we also ran the same set of graphs through `nauty`, `bliss` and `conauto`, and all of these showed much better performance than `Traces` on the large graphs in this collection.

As a sample, we produce in Figure 4 the timing results on the graphs in the package `con_2n` for each of the four isomorphism solvers. The timeout is set at 120 minutes and which can be seen to occur frequently for `Traces`. It should be noted that most of the timeouts occurred due to memory limitations. It seems `Traces` requires large amounts of memory to process these graphs and the swapping required is what leads to the process timing out. While the other solvers (in particular `bliss`) were able to resolve the graphs quickly, they do show fast growth in run times as the graphs get larger. This is explored further in the third set of experiments.

Table 2: Test Environment 2

Feature	Description
Host	B2s Azure VM
Operating System	Ubuntu 18.04 LTS
Memory	8GB
Disk	20GB SSD
CPU	2 CPUs

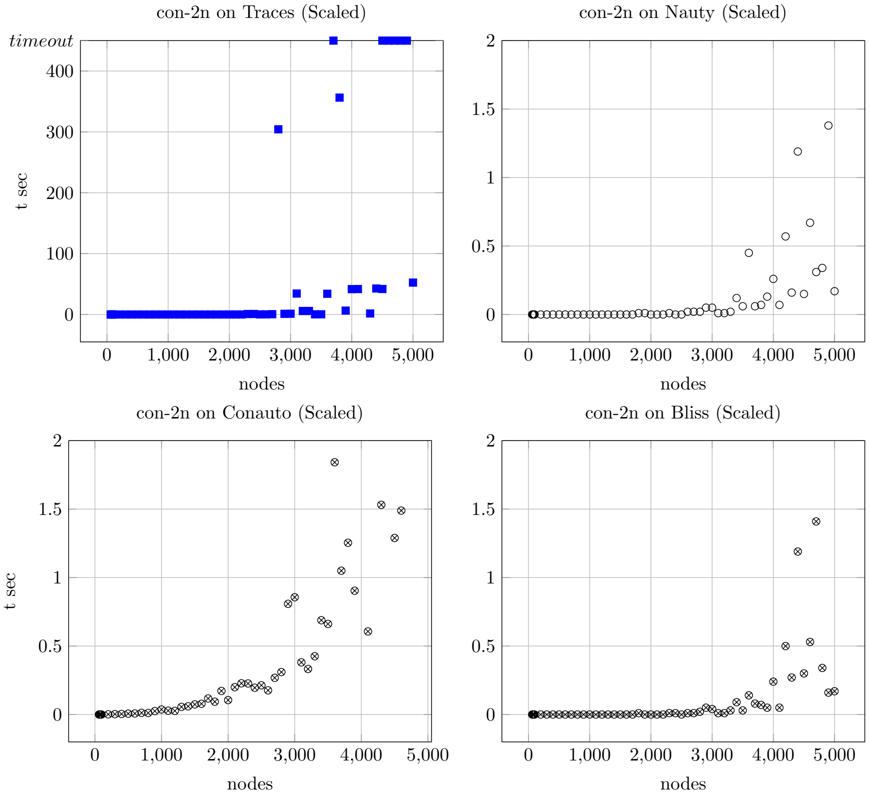

Figure 4: Test run of first set of graphs on four different isomorphism solvers.

Third Set We can report on a third set of experiments performed by Richie Yeung in January-March 2019. The full data on this can be found at `https://github.com/y-richie-y/sat_cfi/`. This generated a new collection of graphs using the same protocol as described in Section 4. These were run again through `Traces`, `nauty`, `bliss` and `conauto`. Graphs with up to 8000 nodes were generated with values of m/n in the range of 1.5-2. Once again, `Traces` frequently (almost invariably) times out on the larger graphs. The performance of the other

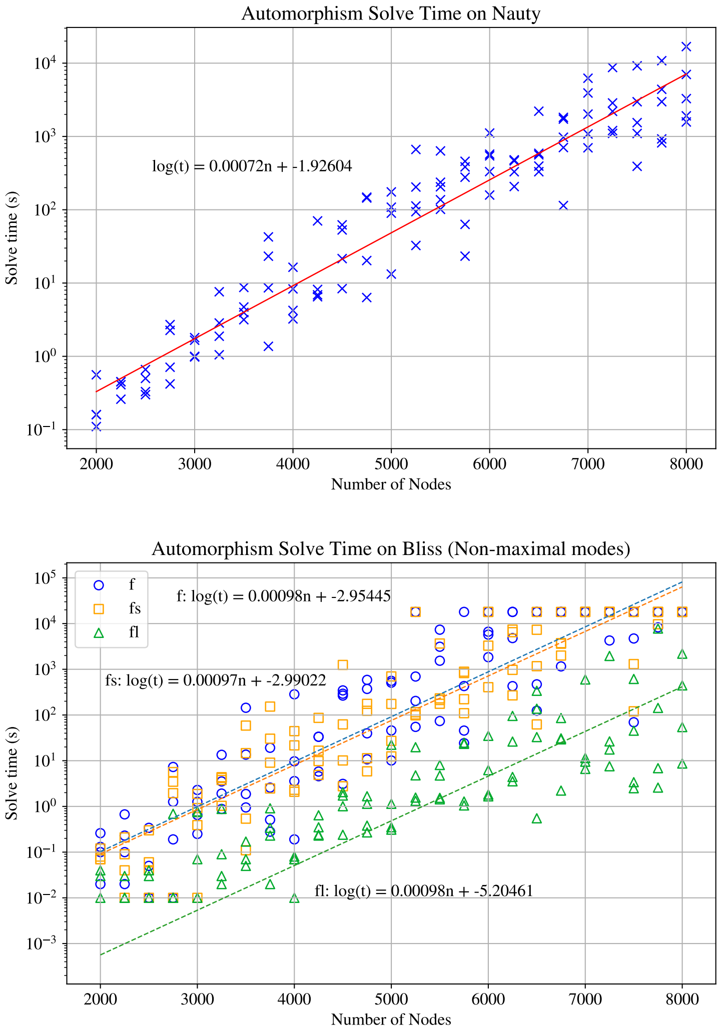

Figure 5: running times on `nauty` and `bliss`

three solvers is better, but exhibits fast growth in running time. For example we exhibit the results for `nauty` and `bliss` in Figure 5, with a logarithmic scale on the y-axis for running time. This is highly indicative of exponential growth in running time. Once again, `bliss` proved to be the fastest of the solvers. However, the performance depends heavily on which target cell selection heuristic is used. As `bliss` allows the use of different heuristics by setting parameters, results three different heuristics are displayed in Figure 5, in different colours. The best performance is for `fl`, which is “first largest non-singleton cell”.

Discussion There are some important points one should highlight from the experimental results. The first is that Traces performs significantly worse on the graphs we construct than any of the other solvers. One possible explanation for this is the fact that the fundamental algorithm in Traces is a breadth-first search procedure of the individualization tree. Such a procedure may require shallowe trees but may, in principle, be more memory-intensive than a depth-first search. An important way that Traces avoids this drawback is the early identification of symmetries in the graph and using this to prune the search space. It is possible that the lack of symmetries in our graphs makes this pruning impossible leading to the solver running out of memory and timing out as a result. The construction described by Neuen and Schweitzer [21] is also aimed at constructing graphs which are asymmetric and have high WL-dimension. They also, similarly, report that Traces is rather slower on their graphs than other solvers. In contrast, Yeung reports that his implementation of the Neuen-Schweitzer construction yields graphs on which Traces performs faster than nauty. This warrants further investigation.

Apart from Traces, an important distinction between the other solvers tested is their cell selection strategy. One of them, bliss, explicitly allows the user to choose the strategy in a call to the system. As we have seen, the choice of strategy can have a significant effect on the performance of the solvers. The results of the third set of experiments, especially on bliss, demonstrate the effect starkly. It would be instructive to understand how these cell selection strategies interact with the construction we have presented.

Our theoretical construction shows that there exists a family of graphs on which any solver based on individualization and refinement, along with factoring by symmetries, will take exponential time, no matter what cell selection strategy is used. Furthermore, it shows that sampling graphs at random from the distribution we describe will produce such graphs with high probability. We cannot verify that the graphs we select do indeed have high WL-dimension, which is why we need experimental validation, and the results do strongly suggest that the growth rate, on any solver, is exponential. In the first set of experiments, we constructed graphs with parameter $m/n \leq 2$ only up to about 5000 nodes. For larger graphs, larger ratios were used. The third set of experimental results extended the construction of graphs with small ratio up to about 8000 nodes (e.g. $n = 800$, $m = 1600$), and the increase in running time is striking. The main reason for using larger ratios to generate the larger graphs in the first case was that at small ratios, finding large 3-XOR formulas that are uniquely satisfiable becomes difficult, requiring large numbers of re-trials with fresh sampling. When this is combining with two runs of a SAT solver for each sampled formula, the time becomes prohibitive. Also, as we are not using the asymmetry gadgets described in Section 3, we are relying on checking that the random 3-left-regular hypergraph we select is itself asymmetric, and this may also involve repeated trials. Here the probability of success decreases with n for a constant ratio. The protocol was improved in the third set by checking unique satisfiability by a direct rank computation. Then, the SAT solver check was only performed for those formulas already known to be uniquely satisfiable, merely to record the difference caused by the use of Gaussian elimination.

6 Conclusion

We have described a theoretical construction of graphs that are provably difficult for a isomorphism solvers such as `nauty` and `Traces`. We have examined the construction experimentally and the results indicate that the graphs produced do indeed show exponential growth rates in running time on these solvers.

The main theoretical result combines known lower bounds for local consistency of 3-XOR formulas with a construction inspired by the graphs of Cai-Fürer-Immerman and the related multipede construction to give a family of graphs which are provably asymmetric and of linear Weisfeiler-Leman dimension. This ensures that the running time grows exponentially with the size of the graphs. Our result also shows that a randomly constructed 3-XOR instance is likely to yield a difficult graph with high probability. That is, the probability tends towards 1 as the graphs get larger.

The experimental setup uses SAT solver technology to create a series of filters which, combined with the random generation of 3-XOR formulas produces graphs to follow the theoretical procedure. For the experimental set-up, we dropped some of the theoretical guarantees on asymmetry and local consistency and replaced them with heuristic tests. This is because we are unable to verify directly the WL-dimension of the graphs constructed.

The results show that our method quickly and consistently produces graphs that are difficult for `Traces`. Experiments with other solvers also support the conclusion that the growth rate of the time taken is exponential. This is comparable with the construction of hard graphs in [21].

Acknowledgements

We would like to thank Richie Yeung for allowing us to report some of his experimental results in this paper. We would also like to thank Jakub Rydval for helping to correct an error in an earlier version of Lemma 1.

References

[1] S. Abramsky, A. Dawar, and P. Wang. The pebbling comonad in finite model theory. In *Proc. of the 32nd IEEE Symp. on Logic in Computer Science (LICS).*, 2017. doi:10.1109/LICS.2017.8005129.

[2] E. Arrighi. Repository of the m1 internship. Git Repository, 2016. https://gitlab.crans.org/arrighi/intershipM1.

[3] A. Atserias. On sufficient conditions for unsatisfiability of random formulas. *J. ACM*, 51:281–311, 2004. doi:10.1145/972639.972645.

[4] A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009. doi:10.1016/j.tcs.2008.12.049.

[5] A. Atserias and A. Dawar. Definable inapproximability: New challenges for duplicator. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 7:1–7:21, 2018. doi:10.4230/LIPIcs.CSL.2018.7.

[6] A. Atserias, J. K. Fichte, and M. Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *CoRR*, abs/1401.3868, 2014. arXiv:1401.3868.

[7] L. Babai. Automorphism groups, isomorphism, reconstruction. In R. L. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics (Vol. 2)*, pages 1447–1540. MIT Press, 1996.

[8] L. Babai. Graph isomorphism in quasipolynomial time. arxiv 1511.03547 v.2, 2015.

[9] L. Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proc. 48th Annual ACM SIGACT Symp. Theory of Computing, STOC*, pages 684–697, 2016. doi:10.1145/2897518.2897542.

[10] L. Babai, P. Erdös, and S. M. Selkow. Random graph isomorphism. *SIAM J. Comput.*, 9:628–635, 1980. doi:10.1137/0209047.

[11] G. V. Balakin, V. F. Kolchin, and V. Kokhlov. Hypercycles in a random hypergraph. *Discrete Math. Appl.*, 2:563–570, 1992.

[12] P. Beame, R. M. Karp, T. Pitassi, and M. E. Saks. The efficiency of resolution and davis–putnam procedures. *SIAM J. Comput.*, 31:1048–1075, 2002. doi:10.1137/S0097539700369156.

[13] E. Ben-Sasson and A. Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48:149–169, 2001. doi:10.1145/375827.375835.

[14] C. Berkholz. Lower bounds for existential pebble games and k-consistency tests. In *Proce. of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 25–34, 2012. `doi:10.1109/LICS.2012.14`.

[15] B. Bollobás. *Random Graphs.* Cambridge University Press, 2nd edition, 2001.

[16] J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. `doi:10.1007/BF01305232`.

[17] Y. Gurevich and S. Shelah. On rigid structures. *Journal of Symbolic Logic*, 61:549–562, 1996. `doi:10.2307/2275675`.

[18] L. Hella. Logical hierarchies in PTIME. *Information and Computation*, 129:1–19, 1996. `doi:10.1006/inco.1996.0070`.

[19] W. Hodges. *Model Theory.* Cambridge University Press, 1993.

[20] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. `doi:10.1016/j.jsc.2013.09.003`.

[21] D. Neuen and P. Schweitzer. Benchmark graphs for practical graph isomorphism. In *25th Annual European Symposium on Algorithms, ESA*, pages 60:1–60:14, 2017. `doi:10.4230/LIPIcs.ESA.2017.60`.

[22] D. Neuen and P. Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 138–150, 2018. `doi:10.1145/3188745.3188900`.

[23] B. Pittel and G. B. Sorkin. The satisfiability threshold for k-XORSAT. *Combinatorics, Probability & Computing*, 25:236–268, 2016. `doi:10.1017/S0963548315000097`.

[24] J. Torán. On the hardness of graph isomorphism. *SIAM J. Comput.*, 33:1093–1108, 2004. `doi:10.1137/S009753970241096X`.